Grouped Queries Indexing For Relational Database

Radosław Boroński
Dept. of Electronics and Computer Science
Koszalin University of Technology
Koszalin, Poland
radoslaw.boronski@tu.koszalin.pl

Grzegorz Bocewicz
Dept. of Electronics and Computer Science
Koszalin University of Technology
Koszalin, Poland
bocewicz@ie.tu.koszalin.pl

Robert Wójcik
Institute of Computer Engineering, Control and Robotics
Wrocław University of Technology
Wrocław, Poland
robert.wojcik@pwr.wroc.pl

Abstract—This paper discusses the problem of minimizing the response time for a given database workload by a proper choice of indexes. We propose to look at the database queries as a group and search for good indexes for the group instead of an individual query. We present condition for applying the concept of grouped queries index selection. Such condition is illustrated by three practical examples.

Keywords—database; index; ISP; grouped queries; related queries

I. INTRODUCTION

Getting database search result quickly is one of the crucial optimization problems in a relational database processing. The major strength of relational systems is their ease of use. Users interact with these systems in a natural way using nonprocedural languages that specify what data are required, but do not specify how to perform the operations to obtain those data [8]. Online Internet shops, analytics data processing or catalogue search are examples of structures where data search must be processed as quick as possible with minimal hardware resources involved. Common practice is to minimize the database search process at minimal cost. A database administrator (or a user) may redesign the physical hardware structure or reset the database engine parameters, or try to find suitable table indexes for a current query. Most vendors nowadays offer automated tools to adjust the physical design of a database as part of their products to reduce the DBMS’s total cost of ownership [3]. As adding more CPUs or memory may not always be possible (i.e. limited budget) and maneuvering within hundreds of database parameter may lead to a temporary solution (wrong settings for other database queries), index optimization should be considered as being foremost.

Indexes are optional data structures built on tables. Indexes can improve data retrieval performance by providing a direct access method instead of the default full table scan retrieval method [7]. In the simple case, each query can be answered either without using any index, in a given answer time or with using one built index, reducing answer time by a gain specified for every index usable for a query [14].

Hundreds of consecutive database queries together with large amount of data involved lead to a very complex combinatorial optimization problem. Two sample tables in a data warehouse of an international automobile factory contain over 1 billion records each (Fig. 1). Time needed to obtain result of both index-less tables joined together may be up to 45 minutes. Such delays are not acceptable for production environment processes. Indexes in such cases may reduce the response time of 50% (depending on which columns are used for the indexing). The classic index selection method focuses on a tree data structure, which could limit the search area as much as possible. Literature acknowledges us with such B-tree types as:

- Sorted counted B-trees, with the ability to look items up either by key or by number, could be useful in database-like algorithms for query planning [5],
- Balanced B*-tree that balances more neighboring internal nodes to keep the internal nodes more densely packed [12],
- Counted B-trees with each pointer within the tree and the number of nodes in the subtree below that pointer [19].

The B-tree and its variants have been widely used in recent years as a data structure for storing large files of information, especially on secondary storage devices [11]. The guaranteed small (average) search, insertion, and deletion time for these structures makes them quite appealing for database applications.

The topic of current interest in database design is the construction of databases that can be manipulated concurrently and correctly by several processes. In this paper, we discuss a simple variant of the B-tree (balanced B*-tree, proposed by Wedekind [20] especially well-suited for use in a concurrent database system [15].

![Figure 1](image-url) Example of large number of rows for two data warehouse tables